Czym jest inżynieria oprogramowania?

Termin „inżynieria oprogramowania” został użyty po raz pierwszy w 1967 roku. Wtedy to odbyła się pierwsza konferencja na ten temat zorganizowana przez NATO. Od tego czasu inżynieria oprogramowania przeszła znaczący rozwój. W niniejszym podręczniku przedstawimy podstawy tego obszaru informatyki technicznej.

Ogólnie można powiedzieć, że inżynieria oprogramowania zajmuje się inżynier skim podejściem do tworzenia oprogramowania. Inżynieria oprogramowania dąży do ujęcia działań związanych z budową programów komputerowych w ramy ty powe dla innych dziedzin inżynierii. W szczególności inżynieria oprogramowania stosuje wiedzę naukową, techniczną oraz doświadczenie w celu projektowania, implementacji, walidacji i dokumentowania oprogramowania.

Inżynierię oprogramowania można podzielić na kilka zasadniczych dyscyplin. Dys cypliny te są związane z typowymi etapami działań inżynierskich w dowolnej dzie dzinie inżynierii. Oczywiście, dyscypliny inżynierii oprogramowania mają istotne cechy wyróżniające je spośród dyscyplin sformułowanych ogólnie. Dla ustalenia uwagi spróbujemy porównać niektóre dyscypliny inżynierii oprogramowania z dyscy plinami inżynierii budowlanej.

Dyscyplina 1. Wymagania

Aby zbudować dom, przyszli jego właściciele lub mieszkańcy muszą określić swoje potrzeby. Te potrzeby należy sformułować tak, aby wykonawcy domu byli w stanie jak najlepiej je spełnić. Potrzeby mogą dotyczyć funkcjonalności domu (układ funkcjonalny, liczba pokoi itp.), jak i jego cech niefunkcjonalnych (estetyka, kolor itp.).

Podobnie w przypadku budowy systemu oprogramowania należy zebrać dokładne wymagania od jego przyszłych użytkowników. Zadanie to jest znacznie bardziej złożone od zebrania wymagań na dom lub nawet złożony budynek wielofunkcyjny. Systemy informatyczne realizują coraz bardziej złożoną funkcjonalność dla coraz większej liczby grup użytkowników. Stąd też bardzo istotne jest uzyskanie od tychże użytkowników (jak również od innych grup zainteresowanych systemem) wszystkich niezbędnych informacji dotyczących ich potrzeb Należy też zwrócić uwagę na to, że potrzeby użytkowników systemów oprogramowania są bardzo zmienne, co zdecydowanie odróżnia te potrzeby od potrzeb właścicieli domów.

Dyscyplina 2. Projektowanie

Na podstawie wymagań zebranych od klientów architekt przystępuje do projektowania domu. Najpierw musi określić strukturę domu, w szczególności z jakich pomieszczeń będzie się składał dom, jakie będą przejścia między pomieszczeniami, ile będzie miał kondygnacji, jak będzie ukształtowany dach itd. Potem ekipa inżynierów konstruktorów projektuje detale konstrukcyjne: typy stropów, rodzaj belek konstrukcyjnych, rodzaj elewacji itd.

System oprogramowania też wymaga projektu. Jest to o tyle istotne, że liczba ele mentów, z jakich skonstruowany jest typowej wielkości system oprogramowania, znacznie przekracza liczbę elementów konstrukcyjnych budynku. Niezbędne jest określenie przez architektów, z jakich składników będzie zbudowany system i na jakich maszynach (komputerach, procesorach) będzie on wykonywany. Bardzo

istotne jest też pokazanie wykonawcom, w jaki sposób system będzie realizował swoją funkcjonalność. Podobnie jak w przypadku projektowania budynków, konieczne jest użycie „rysunków technicznych". Dla systemów oprogramowania istnieje stan dardowy język graficznego opisu systemu, który zostanie przedstawiony w dalszych rozdziałach.

Dyscyplina 3. Implementacja

Plany architektoniczne i projekt konstrukcyjny budynku są podstawą do rozpoczęcia budowy. Zgodnie z projektem należy wytyczyć budynek, zbudować fundamenty, wznieść mury, pokryć dachem i wykonać prace instalacyjno-wykończeniowe.

Wykonanie (implementacja) systemu oprogramowania również wymaga zachowa nia zgodności z projektem. System oprogramowania jest konstruowany poprzez pisanie programów w odpowiednich językach programowania. W zachowaniu zgodności z projektem mogą pomóc odpowiednie metody przekształcania projektu w kod systemu. Dla dobrze zaprojektowanego systemu prace związane z implementacją (kodowaniem) sprowadzają się do uzupełnienia elementów kodu wygenerowanych na podstawie projektu szczegółowego. W niniejszym podręczniku nie będziemy omawiać zasad programowania i języków programowania. Podamy natomiast zasady zgodności kodu z projektem.

Dyscyplina 4. Walidacja

W trakcie i po zakończeniu budowy budynku przeprowadzane są niezbędne sprawdzenia (kontrola jakości). Geodeci sprawdzają, czy budynek został postawiony w prawidłowym miejscu. Służby instalacyjne sprawdzają prawidłowość zainstalowania sieci i urządzeń instalacyjnych. Najważniejsze jednak, aby sami mieszkańcy sprawdzili, czy zostały zrealizowane ich wymagania. W razie wykrycia usterek należy dokonać niezbędnych poprawek. Przy tym należy uważać, aby usterki wykryć w odpowiednim czasie. Jeżeli na przykład stwierdzimy, że przecieka sieć hydrauliczna po wy konaniu wszystkich tynków i podłóg, znalezienie usterki stanie się bardzo kosztowne (trzeba rozkuwać ściany i podłogi). Jeszcze trudniejsze byłoby przesuwanie ścian, jeśli właściciel stwierdzi, że jego potrzeby dotyczące rozkładu pomieszczeń zostały niewłaściwie zrozumiane.

W przypadku systemów oprogramowania wykrycie usterek jest bardzo złożonym problemem. Wynika to przede wszystkim ze złożoności i zmienności wymagań oraz złożoności systemów. Przede wszystkim należy sprawdzić, czy system został zbudowany zgodnie z potrzebami klienta i użytkowników. Można zauważyć, że w zasadzie jedynym kryterium walidacji jest zgodność z tymi potrzebami. System dobrej jako ści to system zgodny z nałożonymi na niego wymaganiami. W trakcie walidacji sprawdza się zatem, czy funkcjonalność systemu jest odpowiednia dla potrzeb klienta. Równocześnie sprawdza się inne elementy jakości, takie jak na przykład niezawodność (w tym odporność na awarie sprzętu) czy wydajność (szybkość działania). Należy podkreślić, że walidacja nie powinna być wykonywana dopiero po zakończeniu implementacji. Powinno się ją przeprowadzać jak najwcześniej w celu zmniejszenia ryzyka niepowodzenia. Podobnie jak w przykładzie z naprawą instalacji, często zdarzają się sytuacje, kiedy niewykryte usterki systemu są bardzo kosztowne w naprawie, jeśli zostały dostrzeżone zbyt późno.

Dyscyplina 5. Nadzór

Podczas projektowania i budowy domu obowiązują pewne procedury związane ze sposobem wykonywania czynności, przepisami administracyjnymi i bezpieczeństwem pracy. Odpowiedni inspektorzy i kierownicy budowy dostosowują te proce dury do warunków na budowie oraz kontrolują ich przestrzeganie.

W trakcie budowy systemu oprogramowania też bardzo istotne jest przestrzeganie pewnych procedur i reguł postępowania. W tym celu zostały opracowane odpowiednie standardowe metodyki. Odejście od przestrzegania (lub po prostu brak) metodyk jest bardzo częstą przyczyną niepowodzeń projektów, w które wkrada się chaos organizacyjny. Czynności zawarte w metodykach dotyczą wszystkich etapów budowy systemu — od wymagań aż do walidacji i oddania systemu do użytku. Nad przestrzeganiem metodyki powinien czuwać odpowiedni kierownik (tzw. metodyk).

Dyscyplina 6. Środowisko pracy

Podczas projektowania i budowy domu bardzo istotne jest środowisko pracy. Współcześni architekci używają systemów CAD (ang. Computer Aided Design), które wyręczają ich w żmudnych pracach kreślarskich. Ekipy budowlane używają odpowiednich narzędzi, które również przyspieszają ich pracę.

Budując system oprogramowania, również powinniśmy bardzo uważnie podejść do budowy środowiska pracy. Bardzo istotny jest wybór narzędzi wspomagających projektowanie systemu. Podczas budowy systemu niezbędne jest posiadanie narzędzi umożliwiających zarządzanie złożonym kodem (dobre środowisko zintegrowane oraz narzędzia do zarządzania konfiguracją i zmianami). Dobrze dobrane środowi sko pracy ułatwia również rozbudowę systemu (tzw. ewolucję systemu). W następ nych rozdziałach opiszemy najczęściej używane narzędzia.

Podstawowe problemy inżynierii oprogramowania

Typowe współczesne systemy oprogramowania składają się z setek tysięcy lub nawet milionów wierszy kodu w różnych językach programowania. Na przykład rozmiar systemu operacyjnego Red Hat Linux 7.1 szacowany jest na 30 milionów wierszy kodu źródłowego. Przekłada się to na jeszcze większą liczbę poszczególnych instrukcji. Jest oczywiste, że takiej liczby elementów nie jest w stanie opanować nawet najbardziej uzdolniony programista. Konieczne jest zatem zastosowanie metod opanowania tej złożoności.

Skąd wynika wspomniana złożoność? W pierwszej kolejności powinniśmy zdać so bie sprawę z tego, że współczesne systemy oprogramowania muszą zapewniać ich użytkownikom bardzo szeroki zakres funkcjonalności. Dla przykładu typowy, zaa wansowany procesor tekstu dostarcza dziesiątki opcji menu, wiele operacji sterowanych klawiszami funkcyjnymi, wiele formularzy i okienek do wprowadzania danych. Taki procesor jest składnikiem większego pakietu aplikacji biurowych (m.in. arkusz kal kulacyjny, narzędzie prezentacyjne). Innym powszechnie znanym przykładem jest system dostępu do usług bankowych za pośrednictwem internetu. Zauważmy, jak wiele opcji zawiera taki system, ile różnych operacji można wykonać (np. dokonać przelewu, zdefiniować odbiorców, wykonać zestawienie transakcji, aktywować kartę bankową, zmienić kod PIN itd., itd.).

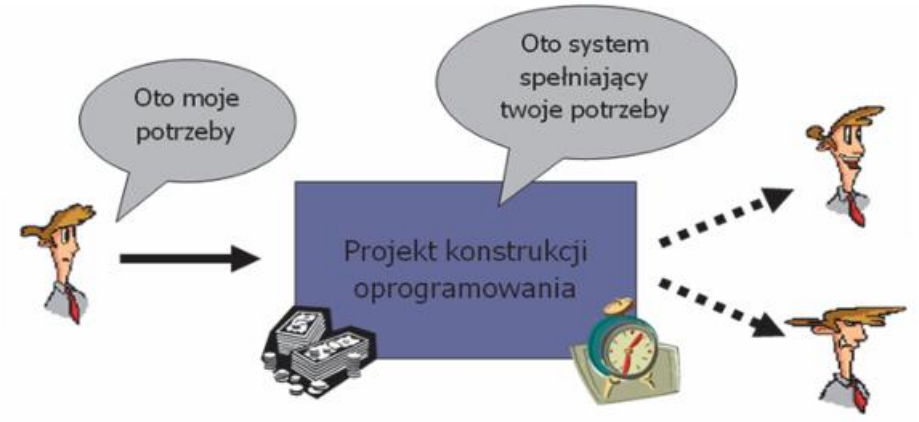
Systemy oprogramowania są spotykane praktycznie we wszystkich współczesnych systemach ułatwiających nam życie. Dotyczy to zarówno systemów, do których mamy dostęp za pośrednictwem komputerów osobistych, jak i systemów nadzorujących pracę różnego rodzaju urządzeń (od maszynki

do golenia, poprzez samolot pasa żerski, aż do statku kosmicznego). Z tego punktu widzenia możemy zatem podzielić systemy oprogramowania na:

1. **Oprogramowanie aplikacyjne**. Są to systemy działające na typowych komputerach, tabletach czy smartfonach, wspierające różnego rodzaju czynności wykonywane w domu i w pracy. W szczególności mogą to być rozwiązania wspierające pracę organizacji biznesowych (od małych firm handlowych aż do dużych korporacji międzynarodowych). Mogą one działać w różnych dziedzinach gospodarki: przemyśle wytwórczym, handlu, telekomunikacji, usługach finansowych (np. bankowość, ubezpieczenia). Mogą to być również systemy do pracy indywidualnej lub do zastosowań w zakresie rozrywki
2. **Oprogramowanie wbudowane**. Są to systemy oprogramowania kontrolujące pracę urządzeń mechanicznych i/lub elektronicznych. Działają one w czasie rzeczywistym, tzn. reagują na sygnały i wymagają czasu reakcji dostosowanego do ograniczeń narzuconych przez dane urządzenie. Na przykład system kontrolujący tor lotu rakiety wymaga korekty lotu tej rakiety na podstawie obliczeń. Korekta musi być wykonana w odpowiednim czasie, gdyż w przeciwnym razie rakieta może ulec uszkodzeniu. Innym przykładem może być system sterowania telewizorem, który wymaga reakcji na odpowiednie polecenia wydawane przez widza.

Złożoność i różnorodność systemów oprogramowania rodzi wiele problemów, które trapią projekty konstrukcji takich systemów. Rozważmy na przykład następujący scenariusz. Zaczyna się kolejne spotkanie komitetu sterującego projektu. Członkowie komitetu z ramienia odbiorcy zgłaszają szereg zastrzeżeń dotyczących jakości dostarczanego systemu oprogramowania oraz niedotrzymywania terminów. Członkowie komitetu z ramienia dostawcy narzekają na kolejną porcję zmian zgłoszonych przez odbiorcę. Projekt zbliża się do końca, a testy akceptacyjne wykazują kompletny brak zgodności produktu z wymaganiami. Projektanci i programiści toczą heroiczne boje o dotrzymanie jakiegokolwiek terminu i zrealizowanie choć części wymagań odbiorcy.

Jest to, niestety, dosyć typowy scenariusz. Różnorodne problemy dotykające współczesne systemy oprogramowania prowadzą do wielu niepowodzeń w budowie tychże systemów. Z punktu widzenia uczestników projektu budowy oprogramowania można wyróżnić trzy kryteria warunkujące powodzenie, co zostało zilustrowane na rysunku poniżej

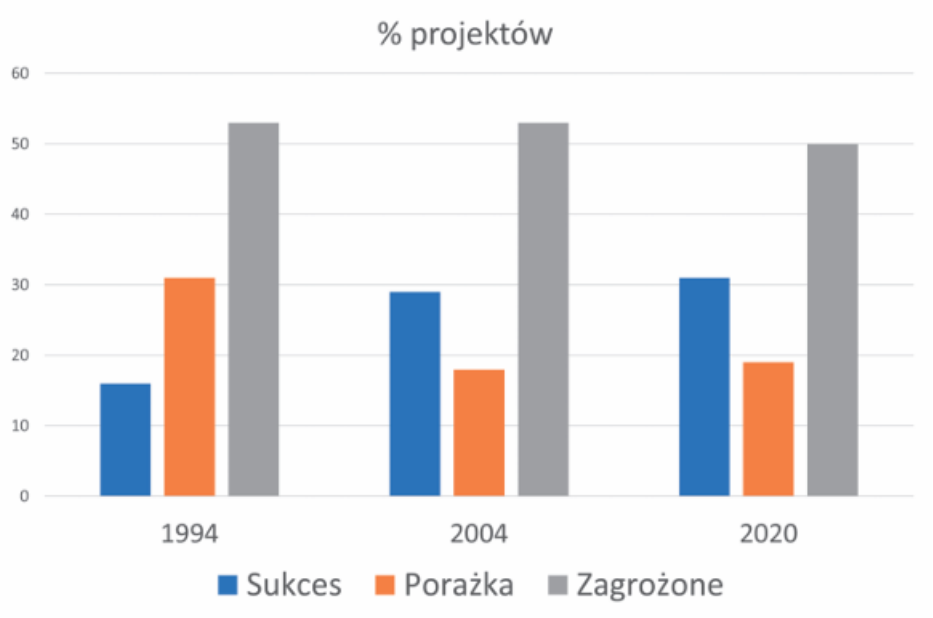


Kryteria te najczęściej określane są na etapie zawierania umowy na budowę systemu. W takiej umowie wskazuje się wymagania dotyczące systemu, termin oddania systemu do użytku i koszt jego wykonania. Można zatem uznać, że projekt skończył

się sukcesem, jeżeli:

- Projekt mieści się w budżecie.
- Projekt mieści się w ramach czasowych (termin zakończenia nie jest przekroczony).
- Dostarczony system spełnia rzeczywiste potrzeby klienta (zadowolony klient).

Niestety, bardzo trudno jest uzyskać spełnienie tych trzech kryteriów. Można tutaj przytoczyć statystyki opracowywane co roku w tzw. raporcie chaosu (ang. chaos report). Rysunek 2 przedstawia wyniki dla lat 1994, 2004 i 2020. Statystyki te pokazują bardzo niekorzystną sytuację w przemyśle wytwarzania oprogramowania. Oznaczają one, że znaczący odsetek projektów (71% w 2004 roku, 69% w 2020 roku) kończy się niepowodzeniem (porażką lub poważnymi problemami z dostarczeniem systemu). Oznacza to olbrzymie straty szacowane na miliardy dolarów/euro/złotych rocznie.



Dodatkowo warto przytoczyć następujące wyniki z 2004 roku:

- 29% — odsetek projektów konstrukcji oprogramowania zakończonych sukcesem.
- 82% — średnie przekroczenie budżetu projektu w stosunku do planu.
- 52% — odsetek spełnienia wymagań.

Można zatem powtórzyć za podstawową tezą pierwszej konferencji NATO nt. inżynierii oprogramowania: „jest kryzys”. Ponieważ jednak stan ten utrzymuje się już od kilkudziesięciu lat, należałoby raczej powtórzyć za Rogerem Pressmannem (Praktyczne podejście do inżynierii oprogramowania, WNT, 2004): „to chroniczna choroba”. Choroba ta ma charakterystyczne symptomy, po których można łatwo poznać, czy również nasz projekt jest nią „zarażony”.

Poniżej przedstawiono kilka przykładowych objawów:

1. Niezadowoleni klienci:

    a. „Nie taki system mieliśmy na myśli”.

    b. „Żaden z naszych urzędników nie będzie w stanie się tego nauczyć”.

c. „Dlaczego ten system działa inaczej niż poprzedni?”.

2. Niezadowoleni dostawcy:

a. „Czy oni sądzą, że tę dodatkową funkcjonalność dostarczymy w tym samym czasie?”.

b. „Dlaczego ciągle zmieniają zdanie? Przecież ta zmiana jest olbrzymia”.

c. „Tak się napracowaliśmy, może i jest brzydkie, ale działa”.

3. Kłótnie o zakres:

a. „Chcecie, żebyśmy zbudowali system dwa razy większy niż zapisano w umowie”.

b. „Ale ta funkcjonalność miała być dostarczona dopiero w następnej wersji”.

c. „Nie dostarczyliście funkcjonalności zapisanej w paragrafie 24, punkt 5a umowy”, „Ależ skąd — dostarczyliśmy”.

4. Chaotyczne zarządzanie zmieniającymi się wymaganiami:

a. „No, proszę państwa, dodajcie tutaj taką małą tabelkę w środku ekranu. To wam zajmie tylko chwilkę”.

b. „Myśleliśmy, że ta zmiana nie będzie dla was tak istotna”.

5. Programiści pracujący w trybie 24/7:

a. „Przywieźcie nam pizzę o północy i dajcie dużo napojów energetyzujących”.

b. „Dajcie nam więcej programistów” (nowi programiści — nowe problemy?).

6. Stres pod koniec projektu:

a. „Ten system pracuje w żółwim tempie”.

b. „Ale jeszcze nie sprawdziliśmy, czy działa ponad połowa funkcjonalności”.

7. Brak stabilności rezultatów między projektami:

a. „To ile tym razem przekroczymy budżet?”.

b. „A jak mam napisać te wymagania?” (to właściwie jak pisaliśmy je ostatnio?).

8. System „prawie gotowy”:

a. „System był już w 90% gotowy, a tu okazało się, że jest jeszcze tyle do zrobienia”.

b. „Oto gotowy system, z tym że poprawa jego wydajności wymaga jeszcze pół roku pracy”.